NeoFi Python Backend

Task Description:

Develop a RESTful API for a simple note-taking application. The API should allow users to perform basic CRUD operations (Create, Read, Update, Delete) on notes.

Requirements:

1. **Endpoints**: Implement the following endpoints:
    - `POST /login: Create a simple login view`
    - `POST /signup: Create a single user sign up view`
    - `POST /notes/create`: Create a new note.
    - `GET /notes/{id}`: Retrieve a specific note by its ID.
    - `POST /notes/share: Share the note with other users.`
    - `PUT /notes/{id}`: Update an existing note.
    - `GET /notes/version-history/{id}:` GET all the changes associated with the note.
2. **Data Model**: Design an efficient schema that can support all the above functions. Include a user model,
3. **Validation**: Implement basic input validation for creating and updating notes. Ensure that required fields are provided and have appropriate data types.
4. **Error Handling**: Handle errors gracefully and return meaningful error responses with appropriate HTTP status codes.
5. **Testing**: Write unit tests to ensure the functionality and integrity of the API endpoints.

Note: Frontend is not required! The task will be evaluated using API endpoints and their response. Feel free to use Django or Flask.

Note:

- Focus on the backend functionality; the UI is not required. You can use tools like Postman or cURL to test the APIs. (bonus points if testing is automated)
- Use appropriate authentication and authorization mechanisms for user registration and login.
- Updating an existing shared note needs some thought input.
- Follow Django best practices, such as using Django's built-in user authentication system and using serializers for API data validation.
- Implement necessary error handling and provide appropriate responses for different scenarios.

Time Limit: 72 hours

Please let me know if you need any further assistance or clarification. Good luck with your project!

Here's a detailed description of the functionality of the API endpoints for the chat application:

1. **User Registration: Endpoint: POST /signup**

Functionality: Allows users to create an account by providing necessary information such as username, email, and password.

Output: If the registration is successful, return a success message or status code.

If there are validation errors or the username/email is already taken, return appropriate error messages or status codes.

2. **User Login: Endpoint: POST /login**

Functionality: Allows users to log in to their account by providing their credentials (username/email and password).

Output:

If the login is successful, return an authentication token or a success message with the user details.

If the credentials are invalid, return an error message or status code

3. **Create new note: Endpoint:POST /notes/create**

Functionality: Create a new note, only authenticated users can create a new note.

Note: note owner needs to be stored in the db. Because notes are shareable.

Output:

If the request is valid, return a success message with status code.

If the request is invalid, return an error message or status code

4. **Get a note: Endpoint:GET /notes/{id}**

Functionality: GET a note by its id. Only authenticated users. A note is viewable by its owner and the shared users only.

Output:

If the request is valid, return a success message with the note content.

If the request is invalid, return an error message with status code

5. **Share a note: Endpoint:POST /notes/share**

Functionality: Share a note with other users. You can parse multiple users in this request. Once the note admin executes this POST api, the users embedded in the request body will be able to view and edit the note.

Output:

If the request is valid, return a success message with the appropriate status code.

If the request is invalid, return an error message or status code

6. **Update a note: Endpoint:PUT /notes/{id}**

Functionality: The note will be editable by admin, and all the shared users. All the users who have access to the note will be able to perform an edit anywhere on the note. For the sake of simplicity, let's assume no existing sentences can be edited. But new sentences can be added in between existing lines of the note. All the updates to the notes need to be tracked with a timestamp, and stored somewhere.

Output:

If the request is valid, return a success message with the appropriate status code.

If the request is invalid, return an error message or status code

7. **Get note version history: Endpoint:GET notes/version-history/{id}**

Functionality: Accessible by users having access only. GET the version history of the note. This includes all the changes made to the note, since it has been created. The response will contain a list of timestamp, user who made the change, and the changes made to the note since its creation. If possible you can track the line number of change as well.

Output:

If the request is valid, return the response with the appropriate status code.

If the request is invalid, return an error message or status code

Evaluation Criteria:

- Completeness: Does the API cover all required functionality?
- Correctness: Does the API behave as expected? Are edge cases handled appropriately?
- Code Quality: Is the code well-structured, readable, and maintainable?
- Asynchronous Implementation: Are asynchronous patterns used where appropriate?
- Error Handling: Are errors handled gracefully with meaningful responses?
- Documentation: Is the API well-documented and easy to understand?
- Testing: Are there sufficient unit tests to ensure reliability?

Submission:

The project should be uploaded on Github and the Git link should be submitted. Apart from that, the repository should have step by step instructions on how to run the project.